OR Spectrum (2009) 31:745–758
DOI 10.1007/s00291-008-0140-9

REGULAR PAPER

k-Shortest routing of trains on shunting yards

Jan Riezebos · Wout van Wezel

Published online: 29 May 2008

Abstract We consider the problem of designing algorithmic support for k-best routing decisions in train shunting scheduling. A study at the Netherlands Railways revealed that planners like to interact with the solution process of finding suitable routes. Two types of interaction were required: the possibility of assigning specific tracks to a route and of preventing the assignment of specific tracks to a route. The paper develops insights in the structure of the cost matrix in this k-best optimization problem. These dominance results are used in a two stage k-shortest path algorithm to support this task of the shunting planners. The solution approach determines the optimal sequence of the tracks that manually have been added to the route and determines the k shortest paths in this network. The approach is implemented in a prototype of a support system for shunting planners. The required calculation times for practical instances of the problem with varying numbers of alternative solutions ($k \leq 8$) and intermediate tracks ($m \leq 5$) are between 0.1 and 1.4 s. These calculation times are acceptable to provide adequate support to the planners of these shunting yards.

Supported by the Netherlands Railways, Project "Rintel 4a". We gratefully acknowledge the management and planners of this company. Specifically, we would like to thank Dr. L.G. Kroon and the planners of location Zwolle for their willingness to co-operate.

J. Riezebos (✉) · W. van Wezel
Faculty of Economics and Business, University of Groningen,
P.O. Box 800, 9700 AV Groningen, The Netherlands
e-mail: j.riezebos@rug.nl
URL: http://www.rug.nl/staff/j.riezebos/

W. van Wezel
e-mail: w.m.c.van.wezel@rug.nl
URL: http://www.rug.nl/staff/w.m.c.van.wezel/

Keywords k-Shortest path · Hamiltonian path · Shunting scheduling · Railway planning

0 Introduction

This paper focuses on the problem of routing trains that have to be relocated during the night. We develop and implement a solution approach that supports planners in this task. The main contribution of this paper is the insight we gained in the structure of the cost matrix in this k-best optimization problem. We implemented these insights in a solution approach that allows for interaction with the planner in terms of the tracks that should or should not be included in a routing. The approach minimizes the number of direction changes and manual crossovers (the main objectives of the planner), while taking into account the distance of the routings.

Section 1 discusses the problem of the planner in detail. Section 2 models this problem as a two-stage problem. The first stage is modeled as a directed k-shortest path problem, and the second stage as a directed k-shortest Hamiltonian path problem in a k-shortest distance matrix. The specific structure of the cost matrix in this latter problem is discussed. Section 3 discusses the way we implemented a solution approach in the prototype of the support system and gives attention to the interaction with the planners. Finally, Sect. 4 presents our conclusions.

1 Context of the problem of train routing

We examine the problem of routing sets of self-propelled coaches from a source track to a destination track during a specific time window. A set of one or more connected coaches is denoted as a train. Trains need a single driver for movement. The coaches that belong to the same train arrive in a specific sequence at the same moment in time on one of the tracks at the shunting yard. Some trains have to leave the station from the same track in exactly the same configuration, but many trains have to be reconfigured or relocated at a different track for some time. Reconfiguration may be necessary for a number of reasons. Firstly, the demand for railway transport fluctuates, so a different number of coaches may be needed in the departing train. Secondly, coaches that arrive at other moments in time or at other tracks may have to be interchanged with coaches in this train, for example because of maintenance schedules of the coaches. Relocation of coaches on the shunting yard may also be necessary for a number of reasons. The track where the train arrived may be used primarily for pass-through traffic, so if it has to stay there during a longer period of time (e.g., a night shift), it has to be relocated to another track on the shunting yard. Next, activities such as washing, cleaning, or inspection, may be required. These activities have to be performed at a specific track where the required facilities are available. The final reason is that there may be a difference between the arrival track and the departure track for this train, which makes relocation necessary.

In our study, we worked on real-life data of the location Zwolle. Figure 1 shows the track layout. The tracks that are encircled in the figure will be used in this article to explain the routing algorithm we developed.

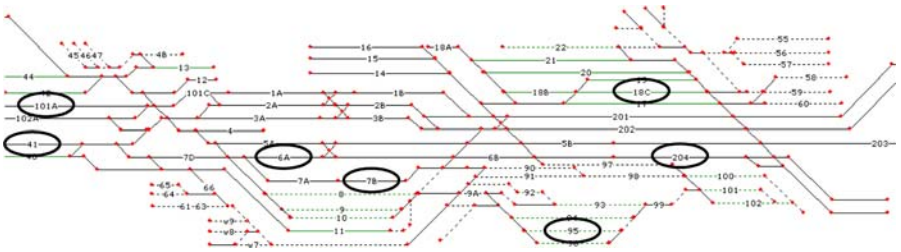

Fig. 1 Track overview

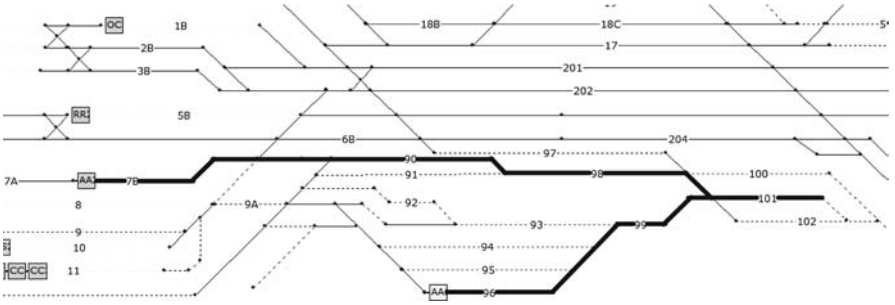

Fig. 2 Route with a saw-movement at track 101

A shunting planner plans the reconfiguration, relocation, and routing of the trains during the night. Planners aim at efficient schedules. Efficiency means first of all minimal walking times for the shunting team and secondly minimal driving times for trains when they are relocated. Inefficiencies occur if tracks are used that are not included in the automatic crossover system, or if a train has to change direction on a track. In the latter case, a saw-movement of a train is needed in order to continue the routing. For example, track 101 in Fig. 2 is used for such a saw-movement in the route of train AA from track 7B to track 96. As the self-propelled trains can be controlled from both the front side and the rear side, drivers realize the saw-movement by driving to a track where the train is allowed to stop, getting off the train, walking to the other side of the train, and starting it again. Hence, the driver has to walk a distance equal to the total length of the train, which makes routes involving a saw-movement inefficient. Moreover, direction changes are only allowed at specific tracks that have to be long enough, as for safety reasons a train may only be halted at tracks that are surrounded by official rail signals. A planner therefore tries to minimize the number of direction changes and usage of manual crossovers in a route. The actual distance of the train on a route is of secondary importance.

As several shunting teams are simultaneously active on the railway network, the planner has to avoid that a collision occurs. The planner uses a standard time window of five minutes for moving a train. This time lag is long enough to cover the saw-movements that may be required and walking time to the next assigned shunting task. Tracks that already have been assigned to other routings that partly will be performed in parallel cannot be included in the current route. Neither can tracks be included at

which already a train has been located during a part of this time window. The set of available tracks for a routing decision is therefore time-dependent and the length of the time window affects the size of the set of available tracks.

The sequence of decision-making that a planner chooses does also affect the contents of the set of available tracks for the decision on the route. For the next routing decision of another train within the same time window, the set of available tracks will be smaller.

Although optimization models have been build for similar problems (see Zwaneveld et al. 1996; Brucker et al. 2002; He et al. 2003; Lübbecke and Zimmerman 2003a; Freling et al. 2005; Dessouky et al. 2006; and for an overview Cordeau et al. 1998), the Netherlands Railways has chosen to not yet implement a model that integrally solves the routing problem in their railway planning systems. Integral approaches often lack important real-world concerns (see, e.g., Cai et al. 1998). Lübbecke and Zimmerman (2003b, p. 699) state that "the planning process can be covered only partly by a model". Alternatively, Carey and Carville (2003) developed an approach that mimics the behavior of a human planner. However, such an approach does not benefit from intelligent algorithmic support. In this paper, we use a more human centered approach of algorithmic design that could help to overcome this type of problems. The approach aims to develop OR-support for small but frequently occurring planning tasks of a human planner. The human planner determines the boundaries of the problem that will be supported with an algorithm, as well as the constraints.

From several interviews with planners, three mechanisms were identified that they would like to use when constructing a route. First, a planner would like to be able to block tracks that should not be included in the route. According to the hard constraints of the problem, many tracks are blocked automatically. However, the planner sometimes likes to block some other tracks as well, for example if these tracks will be needed for yet unplanned movements. Second, the planner would like to be able to specify tracks that should be included in the route. There are two purposes for this functionality. It enables a planner to quickly affect the geographical area of the main part of the route. Next, it enables the planner to identify specific tracks that should become part of the route. The latter functionality provides the planner the possibility of regaining control over the route determination, for example, because of building trust in the computer technology or including characteristics of these tracks that are not included in the model. The algorithm will have to complete the route and determine the optimum order of visiting these tracks. Finally, the planner would like to be able to select a route from a list of equally efficient routes.

2 Modeling and design of solution approach

The problem for which we design an algorithm is now formulated as:

> Given the set $\Im$ of available tracks and the set $\Re \subset \Im$ of tracks that need to be included in the route, find k alternative routes for routing a train from a source track to a different destination track, such that each alternative route includes all m tracks that belong to the set $\Re$.

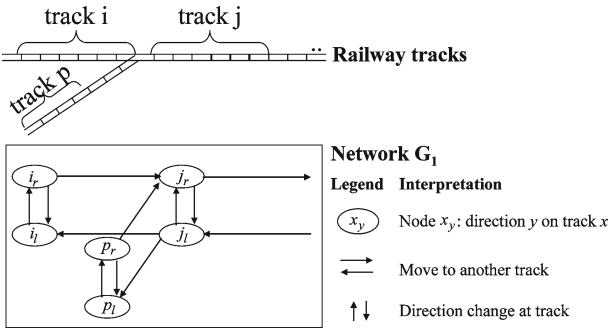

Fig. 3 Initial network G_1 and corresponding railway tracks

Set $\Im$ is time-dependent and lists all tracks that will be available for routing during the next five minutes after starting the routing. The planner may have removed several tracks from this computer-generated list, so at the end both sets $\Im$ and $\Re$ are predetermined by the planner.

Note that this problem formulation uses a time-dependent graph, not a time-expanded graph. The latter type of graph would be necessary if more detailed information on the planned moment of using a track within the five minute time lag should be included.

In the classification of Cordeau et al. (1998), our problem formulation is a compound network routing problem. In order to develop an algorithm for this problem, we have to take into account the direction of moving a train on a track. The shunting yard is therefore modeled as a directed graph G_1 where each track i $\in \Im$ is represented by two nodes depending on the direction the train traverses this track (see lower part Fig. 3). Node i_r points to a right direction, node i_l to a left direction of the train at this track. [Note the notational difference between i (track) and i (node)]. The total set of nodes in graph G_1 is denoted as N.

If an arc connects two nodes that represent the same track, a train can change to that direction on this track. If an arc connects two nodes that represent different tracks, routing a train between these tracks is possible in the direction given. The total set of directed arcs is denoted as V, so $G_1 = \{N, V\}$.

Costs are assigned to the arcs in the network in order to search for a route that minimizes total costs. We model the objective of the planner (discussed in Sect. 1) as to find the set of k shortest routes that avoid manual operation of crossovers and direction changes as much as possible. The costs of the arcs and paths in the network therefore represent the weighted sum of these objectives, where the weight for manual crossovers and direction changes is much higher than the weight for the actual distance. The costs for an arc between nodes i and j are expressed as:

$$c(i,j) = w_1 I_c(j) + w_2 I_d(i,j) + w_3 d(i,j) \quad \forall (i,j) \in V$$

where $c(i,j)$ = cost of arc from node i to node j

w_n = weight of nth cost factor

$I_c(j)$ = 1 if track j requires manual crossover; else 0

$$I_d(i, j) = 1 \text{ if i} = \text{j} \wedge i \neq j \text{ (saw-movement); else 0}$$
$$d(i, j) = \text{physical distance between track i and j.}$$

The weights for manual crossovers and direction changes are chosen such that they are always larger than the sum of all arc costs associated with physical distance, i.e., $w_1 \geq w_2 \geq \sum_{i,j} d(i, j); \quad w_3 = 1$.

Network G_1 can be used in a shortest path algorithm to determine a path from source i^* to sink j^*. Assuming that the shortest path visits nodes $z_1, z_2, \ldots, z_t$ before arriving in node j^*, the path costs are $a_1(i^*, j^*) = c(i^*, z_1) + \sum_{l=2}^{t} c(z_{l-1}, z_l) + c(z_t, j^*)$, where the ordered vector $a(i, j) = (a_1(i, j), \ldots, a_k(i, j))$ contains the costs of the k-shortest paths from node i to j; $i, j \in N$.

In order to find k different routes, a k-shortest path algorithm can be used. Several algorithms are available in literature, mostly based on the path deletion algorithm of Yen (1971) or the labeling algorithm of Shier (1976) (see Eppstein 1998 for an overview). However, there is no guarantee that the routes found will visit all m tracks in the set $\Re$. Therefore, we have to develop a new algorithm in order to find the k shortest routes from source to destination that visit all m intermediate tracks.

We propose a two-stage solution approach. The first stage uses the Shier algorithm to calculate the k-shortest lengths of paths from the source node i^* and each node of the subset $S \subset N$ to the sink node $j^* \notin S \cup i^*$. $S = \{i | i \in N; \text{i} \in \Re\}$ and has cardinality $2m$, with m the cardinality of $\Re$. The second stage uses this information to determine the sequence of visiting the m tracks in each of the k alternative routes from source i^* to sink j^*. The main results of this paper, especially the dominance results in case of k-shortest problems, are being developed in this second stage.

Shier's k-shortest path algorithm (Shier 1976) calculates in a single pass all k-shortest paths from a source node to all other nodes in the network, including the sink node. The running time of the algorithm is $O(kn^3)$, with n the number of nodes in N. We iterate this algorithm $2m + 1$ times to generate the k shortest paths from all elements of $i^* \cup S$ to all nodes in N, including the sink node j^*. This iterative procedure is for the typical problem instances faced by the planners in Zwolle ($n \approx 400$, $k \leq 8$ and $m \leq 5$) quite efficient, while still providing the optimal solution to the problem of stage 1. Stage 1 results in a matrix $A = [a(i, j)]$ of minimal cost routes. Note that as Shier's algorithm distinguishes paths only by length, it cannot save two different paths of the same length, hence $a(i, j)$ is a vector with $a_1(i, j) < \cdots < a_k(i, j)$. In our problem context, different paths with equal lengths between source and sink will be very rare, as the data for the length of tracks is exact. Hence, it is not necessary to accommodate for this characteristic of Shier's algorithm.

Table 1 shows an illustration of the matrix A. For the railway network of Zwolle some of the results are shown for a routing problem with $k = 2$ alternatives and $m = 3$ intermediate tracks. The costs of a manual crossover w_1 and of a direction change w_2 have both been set to 1.000.000, so the set of k shortest paths will focus on minimizing the sum of manual crossovers and direction changes. Table 1 only shows the $2m$ intermediate nodes and the sink node as destinations, although the algorithm calculates the distances to all nodes in the network. We only need this selection of nodes in stage 2 of the solution procedure.

Table 1 Cost matrix A with $k = 2$ alternatives and $m = 3$ intermediate tracks

From/to	k	204_r	204_l	41_r	41_l	$101A_r$	$101A_l$	Sink $6A$
Source $7B$	1	6422	1006422	2012109	1012109	2012158	1012158	1006929
	2	6644	1006644	2012326	1012326	2012167	1012167	1007146
204_r	1	0	1000000	2013199	1013199	2013234	1005927	1008019
	2	∞	1005428	2013420	1013420	2013252	1012879	1008240
204_l	1	1000000	0	1013199	13199	1013234	13234	8019
	2	1005428	∞	1013420	13420	1013252	13252	8240
41_r	1	13199	1013199	0	1000000	2006115	1006115	5180
	2	13420	1013420	∞	1002712	2008247	1008247	5623
41_l	1	1013199	2013199	1000000	0	3006115	2006115	1005180
	2	1013420	2013420	1005916	∞	3008247	2008247	1005623
$101A_r$	1	13234	1013234	2006115	1006115	0	1000000	5242
	2	13252	1013252	2008247	1008247	∞	1005734	1009740
$101A_l$	1	1013234	2013234	3006115	2006115	1000000	0	1005242
	2	1013252	2013252	3008247	2008247	1006746	∞	2009740

The second stage of the solution procedure aims at determining the optimal sequence of visiting the m tracks of the set $\Re$ in each of the k alternative paths from source to sink. This is an interesting problem that to the knowledge of the authors has not been studied before. It is related to the Hamiltonian path problem of finding the optimal sequence of visiting the nodes in a network where each node has to be included *exactly once* (see Chvatal 1985). However, in network G_1, each track $\in \Re$ was represented by two nodes: i_r and i_l that both belong to S, but only one of these nodes needs to be present on a path from source to sink. So we need to determine the nodes that need to be included in the path and the actual sequence of visiting these nodes.

Our solution approach will be to exploit the structure of the specific cost matrix when solving a Hamiltonian path problem in each of the 2^m different networks that emerge, as there are m intermediate tracks. Figure 4 shows the aggregated network G_2, from which the 2^m networks used in stage 2 are deduced. The figure shows an aggregated network in the sense that the intermediate nodes (e.g., i_r *or* i_l) are no real nodes, but aggregate nodes. All other things being equal, each aggregate node is represented by two different networks with either node i_r or node i_l. As this holds true for all m intermediate nodes, a total of 2^m networks results. The complexity of the problem is mainly due to the fact that the k alternative paths in the final solution may result from more than one network in this set of 2^m networks. However, some of them may also result from the same network, either by applying another sequence of visiting these nodes (a next-best Hamiltonian path) or by using a next-best arc with a slightly higher cost on the same path.

Let us first gain more knowledge of the structure of this problem. The 2^m networks that result from G_2 (Fig. 4) differ from G_1 (Fig. 3) in three respects. First, these networks contain only a small subset $i^* \cup S_{\text{sel}} \cup j^*$ of the nodes from G_1, where $S_{\text{sel}} = \left\{ i_x \in S | \, \text{i} \in \Re \wedge i_y \notin S, x \neq y \right\}$, i.e., S_{sel} is a subset of S with cardinality m

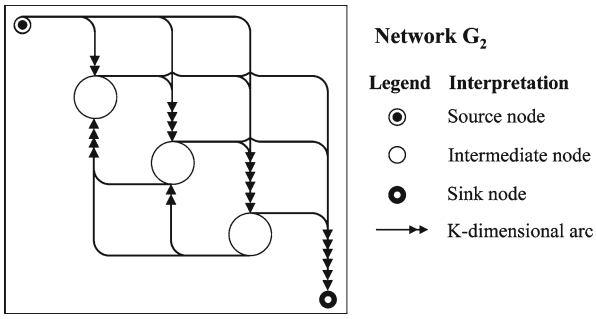

Fig. 4 Aggregated network G_2 of stage two

where each intermediate track i is represented by exactly one node, either i_r or i_l. There are 2^m different subsets S_{sel}, resulting in the same number of networks $G_2\,(S_{\text{sel}})$. Each intermediate node in a network $G_2\,(S_{\text{sel}})$ can be reached from the source node, and each intermediate node can reach the sink node. There are no directed arcs between source node and sink node, unless $S = \emptyset$. If there are intermediate nodes, the network contains between each pair of intermediate nodes a total number of $2k$ directed arcs. The cost of the kth arc between two nodes represents the cost of the kth-shortest path between the corresponding nodes in G_1. Hence, the optimal solution of stage 1 is used as input for the networks of stage 2.

The problem of stage 2 is now to determine the k minimal cost paths from i^* to j^* in the set of 2^m networks $G_2\,(S_{\text{sel}})$, where $G_2\,(S_{\text{sel}})$ can be characterized as:

$$\begin{aligned} G_2\,(S_{\text{sel}}) &= \left(N_2^{\text{sel}}, V_2^{\text{sel}}\right) \text{ with} \\ N_2^{\text{sel}} &= \{i^* \cup S_{\text{sel}} \cup j^*\} \\ V_2^{\text{sel}} &= \{a_h\,(i,\, j)\,|i \in i^* \cup S_{\text{sel}},\, j \in S_{\text{sel}} \cup j^*,\, h \in [1,k]\} \\ &\quad / \{a_h\,(i^*,\, j^*)\,|h \in [1,k]\}\,. \end{aligned}$$

Before we solve this specific shortest path problem, we take a further look at the characteristics of the cost matrix A (see Table 1 for an example). Contrary to the normal situation in a shortest path problem, this cost matrix shows the costs of k different paths between the same nodes. If we aim at finding multiple shortest paths in networks with such a cost matrix, the assumptions behind the cost matrix in normal k-shortest path algorithms (Eppstein 1998) should be reconsidered. We therefore pay attention to the cost matrix A. The inherent structure of A can be revealed using four Lemmas. They hold for all $i \in i^* \cup S_{\text{sel}};\ j \in S_{\text{sel}} \cup j^*$.

Lemma 1

$$a_1(i,\, j) \leq a_1(i,\, z) + a_1(z,\, j) \quad \forall z \in S_{\text{sel}}. \tag{1}$$

Proof Follows directly from the optimality of stage 1.

Lemma 1 states that the triangle equation holds for the best alternative paths between nodes i and j. $a_1\,(i,\, j)$ is the minimal cost of a path between node i and j in G_1. Any node z that is in the subset S_{sel} is also a node in G_1, as $S_{\text{sel}} \subset N$. The optimality of the

k-shortest path algorithm of stage 1 guarantees that the triangle equation also holds for the minimal cost paths between any pair of nodes in G_2. □

Lemma 2

$$a_h(i,j) < a_{h+1}(i,j) \quad \forall 1 \le h < k. \tag{2}$$

Proof Follows directly from using the Shier (1976) algorithm in stage 1, as this algorithm finds k routes with different costs, assuming that these routes exist (i.e., $a_h(i,j) < \infty \forall 1 \le h \le k$).

Lemma 2 states that the matrix A has between each set of nodes k strictly increasing costs of paths. Note that there may be several paths with the same costs. □

Lemma 3

$$\begin{aligned} &IF\ \exists z \in S_{\text{sel}} : a_h(i,j) < a_f(i,z) + a_g(z,j) \quad \text{with } f,g \in [1,k], h \in [1,k-1]. \\ &Then \qquad a_{h+1}(i,j) \le a_f(i,z) + a_g(z,j). \end{aligned} \tag{3}$$

Proof Suppose the contrary would be true: $a_{h+1}(i,j) > a_f(i,z) + a_g(z,j)$. Then [according to (2)] we would have $a_h(i,j) < a_f(i,z) + a_g(z,j) < a_{h+1}(i,j)$, so there would be a path with a cost between $a_h(i,j)$ and $a_{h+1}(i,j)$. However, the optimality of stage 1 guarantees that $a_{h+1}(i,j)$ is the minimal cost path in the network $G_1 = (N,V)$ with a cost that exceeds $a_h(i,j)$. As $S_{\text{sel}} \subset N$, this holds true for $z \in S_{\text{sel}}$. □

Corollary 1

$$\begin{aligned} &IF\ \forall z \in S_{\text{sel}} : a_h(i,j) < a_f(i,z) + a_g(z,j) \quad \text{with } f,g \in [1,k], h \in [1,k-1]. \\ &Then \qquad a_{h+1}(i,j) \le a_f(i,z) + a_g(z,j)\ \forall z \in S_{\text{sel}}. \end{aligned} \tag{4}$$

Lemma 4

$$\begin{aligned} &IF\ \exists z \in S_{\text{sel}}, z \ne i,j : a_h(i,j) = a_f(i,z) + a_g(z,j) \quad \text{with } f,g,h \in [1,k-1]. \\ &Then \qquad a_{h+1}(i,j) \le a_{f+1}(i,z) + a_g(z,j) \\ &and \qquad a_{h+1}(i,j) \le a_f(i,z) + a_{g+1}(z,j). \end{aligned} \tag{5}$$

Proof Suppose $a_{h+1}(i,j) > \min\left[a_{f+1}(i,z) + a_g(z,j), a_f(i,z) + a_{g+1}(z,j)\right]$, i.e., the contrary would be true. Then we would have (using Lemmas 2, 3): $a_h(i,j) < \min\left[a_{f+1}(i,z) + a_g(z,j), a_f(i,z) + a_{g+1}(z,j)\right] < a_{h+1}(i,j)$, but $a_{h+1}(i,j)$ is the minimal cost path in the network $G = (N,V)$ exceeding $a_h(i,j)$. Hence the contrary cannot be true.

Lemmas 3 and 4 show that the triangle equation, which holds for the best solution between node i and j (Lemma 1), does not generally hold for second-best solutions. However, there is still some structure left in the matrix A. This structure is presented

in Lemmas 3 and 4 and can be used as dominance relations in a solution approach for stage 2.

Based on these Lemmas, we developed a heuristic for the second stage problem. The heuristic is based on a Hamiltonian path approximation. A Hamiltonian path in a network $G = (N, V)$ is a path that visits all nodes in N exactly once. In a cost matrix that obeys the triangle equation (see Lemma 1), the search for a path that visits all nodes at least once can be restricted to the search for a Hamiltonian path, i.e., to the search for a permutation of the intermittent nodes such that the minimal cost path between source and sink results. Such a permutation largely restricts the search process, although the Hamiltonian path problem is still NP-complete (Chvatal 1985). Lemma 1 has shown that the cost matrix A does obey the triangle equation for the minimal cost path between each pair of nodes in a network G_2 (S_{sel}), but this cannot be generalized to all k-best cost paths in that network, as Lemmas 3 and 4 have shown. Therefore, the Hamiltonian path solution has to be considered as an approximation of the optimal solution. The latter might contain multiple visits to the same node for alternative solutions, and such solutions will not be found using a Hamiltonian approximation.

We introduce a branch and bound method in order to cope with the selection of networks S_{sel} for which a Hamiltonian path problem is solved. The branch and bound approach aims at determining in an early stage of alternative solution generation whether a solution that starts with an alternative permutation will result in an improvement, using Lemma 1. As bounding mechanism, it uses the reduced distance matrix approach (Balas and Toth 1985). Whenever an alternative solution has been found, it determines which of the k-best solutions found so far are being improved by this new solution, for which Lemmas 2, 3 and 4 are used.

There exists extensive literature on k-best combinatorial optimization problems, see Eppstein's (2001) internet bibliography http://www.ics.uci.edu/~eppstein/bibs/kpath.bib. Solution methods for k-best shortest path and Hamiltonian path problems are given in Lawler (1973) and Hamacher and Queyranne (1985). Sensitivity in k-best traveling salesman problems is discussed in van der Poort et al. (1999). The basic trade-off is between continuing to use the same sequence of nodes (i.e., by using a second-best path between a pair of nodes in this sequence) and switching to another sequence of nodes. In the latter case, we can start with the minimal cost arcs a_1 (i, j) available for this new sequence. This trade-off problem, denoted as the tolerance problem in literature on combinatorial optimization problems (e.g., Shier and Witzgall 1980; Libura 1991), is as difficult to solve as the original combinatorial problem. Due to the relatively small k and m in the practical problem instances that we have to solve, we have decided to not develop a heuristic for this trade-off problem, but solve it optimally using a branch and bound approach that incorporates the Lemmas 2, 3 and 4.

The two stage approach for determining the k-shortest routes on a shunting yard, visiting the list of must-visit tracks at least once, and avoiding the tracks that are listed in the blocking list, is hence as follows:

Stage 1 Let a node represent the direction of a train on a track. Determine k-best cost paths from a subset $i^* \cup S$ (the source node and all $2m$ intermediate nodes) to

all nodes N (including sink node j^*) in the network G_1. Output: all k-best costs $a_1(i, j) \ldots a_k(i, j)$ of paths originating from nodes used in stage 2.

Stage 2 Approximate the optimal solution by finding the k-best Hamiltonian paths in the set of all 2^m networks $G_2(S_{\text{sel}})$, where the nodes in each network represent all tracks that have to be included in a route.

(a) Initialization: For all subsets S_{sel} set $a_1(i, i) = \infty \, \forall i \in S_{\text{sel}}$ (avoid these sub tours)
Set $CurSolution[h] = \infty \forall h \in [1, k]$

(b) For each subset S_{sel} do apply a branch and bound procedure to determine whether paths in this subset will improve the current set of solutions.

If $\sum_{(i,j) \in Hamiltonian\ path} a_1(i, j) < CurSolution[k]$, update the current solutions, using Lemmas 2, 3, and 4.

(Note that the algorithm enumerates all subsets S_{sel}, but does not specify the sequence of checking the subsets. If a path is found that improves one or more elements in the array of current solutions, we apply Lemmas 2, 3 and 4 in order to construct an updated array of k-best solutions.)

The branch and bound method uses a lower bound LB in order to determine whether a branch will lead to a path with a cost that exceeds the current k-best solution found. The lower bound supposes that a partial Hamiltonian path has been created with $i^\bullet$ the last node added, leaving a subset $T \subset S_{\text{sel}}$ of intermediate nodes to be added. It is calculated as follows:

Set $a_1(i^\bullet, j^*) = \infty$ (it is not allowed to go directly to the sink node, as there are other nodes left in T that have to be added to the path)

For all nodes $i \in \{i^\bullet \cup T\}$ calculate $r_i = \min_{j \in \{j^* \cup T\}} [a_1(i, j)]$

For all nodes $j \in \{j^* \cup T\}$ calculate $q_j = \min_{i \in \{i^\bullet \cup T\}} [a_1(i, j) - r_i]$

$$LB = \text{Cost partial path} + \sum_{i \in \{i^\bullet \cup T\}} r_i + \sum_{j \in \{j^* \cup T\}} q_j.$$

If $LB \geq CurSolution[k]$, abandon this branch, else continue the search within this branch.

3 Implementation

This section discusses the way we implemented the algorithm in a prototype of the planning support system and gives attention to the interaction between this system and the planner.

For each train movement a route must be specified. The system contains a graphical overview of the tracks with which routes can be created manually, interactively, or automatically (Fig. 2). The overview shows the tracks of the currently active route bold in a different color. Furthermore, track occupancy and track characteristics such as manual crossover are shown using color and line style.

Table 2 Average calculation times (milliseconds) for routing trains

Number of alternatives k						
Number of obligatory tracks m	0	1	2	3	4	5
Obligatory track:		204	41	101*A*	18*C*	95
$k = 1$	114.2	248.6	424.4	526.9	714.9	936.9
$k = 2$	115.9	251.7	442.7	640.9	880.9	1, 181.3
$k = 3$	126.3	280.8	443.9	636.7	851.3	1, 106.1
$k = 4$	129.2	287.8	475.5	694.5	918.9	1, 190.5
$k = 5$	128.3	296.9	500.0	712.0	913.4	1, 212.0
$k = 6$	126.6	298.3	496.6	707.5	939.9	1, 267.0
$k = 7$	132.2	300.0	509.9	720.5	943.6	1, 275.2
$k = 8$	129.5	333.3	529.8	709.4	964.8	1, 375.0

Average calculation times (milliseconds, 100 replications) on a 2.4 Ghz Intel Pentium 4 processor

The planner can change the route interactively by blocking tracks or making tracks obligatory by clicking on the tracks in the graphical overview. After each action of the planner, the algorithm creates a new set of alternative routes taking the information from the planner as a constraint. The planner can choose from the k alternative tracks that are shown in a window with routing alternatives.

Table 2 shows the algorithm calculation times for a train that must be routed from track 7B to track 6A. The table shows the average times for $k = 1$–8 alternatives and for $m = 0$–5 added obligatory tracks, based on 100 replications. As a benchmark, the average time for the default routing mode ($k = 1$ alternative, $m = 0$ obligatory tracks, a simple shortest path problem) is 0.114 s. This mode is used when the planner is making the shunt schedule and routes are calculated in the background automatically without user intervention. The planner can manually increase the number of alternatives k when he or she would like to see more routes. The table shows that this decision has no large impact on the required calculation times compared to the simple shortest path problem calculation, as long as the number of additional obligatory tracks m is low. The inclusion of additional obligatory tracks results in longer calculation times, but still very acceptable in our type of application. The scenario with 8 alternatives and 5 additional obligatory tracks results in calculation times of less than 1.4 s. Planners of the Netherlands Railways expressed that larger values of k or m were not necessary in practice, which make the calculation times acceptable for the practical application.

4 Conclusions and future research

This paper has developed insights in a k-best routing problem and implemented a solution approach in a prototype of a planning support system. Support is provided to railway planners that perform the task of routing trains on a railway network. Trains have to be relocated to different tracks if the original track is required for other purposes. A planner may prefer to use specific tracks in a route for a train, because of

infrastructural or other reasons. A decision to block usage of several tracks may also be appropriate, e.g., if these tracks will be needed for other purposes. The implementation enables a planner to graphically interact with the solution process and select a route from a set of k alternatives.

The solution approach that we propose is a two-stage process. The first stage consists of determining the k-shortest routes from the "must-visit" tracks (including the source track) to all other tracks in the railway network. We apply a k-shortest path algorithm in a directed graph, where each track is represented by two nodes. The design of the network enables us to determine the direction of the train over the tracks, which is important for reasons of feasibility of the solution.

The second stage consists of determining the sequence of visiting the tracks in the "must-visit" list for any of the k best solutions. We show that this problem differs from traditional k-best Hamiltonian path problems, as the distance matrix consists of k distances for each pair of nodes.

The main results of this paper are in developing insights on the cost matrix of this problem. We develop dominance relations and apply them in a branch and bound solution approach. The result lists the k shortest paths from source to destination that visit all tracks in the "must-visit" list exactly once. The solution approach is implemented in a prototype of a support system for the planner. Some experiments were done with the solution approach, using real-life data. The results of these experiments showed that the generated alternatives were logical and obtained within a reasonably short time (0.1–1.4 s).

Although the solution approach has been developed for a specific routing problem in shunting scheduling, the approach is of relevance for other routing problems as well. *k-best* solution approaches provide opportunities for planners to combine their insights and implicit preferences with the power of operations research methods.

References

Balas E, Toth P (1985) Branch and bound methods for the traveling salesman problem. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds) The traveling salesman problem—a guided tour of combinatorial optimization. Wiley, Chichester, pp 361–401

Brucker P, Heitmann S, Knust S (2002) Scheduling railway traffic at a construction site. OR Spectr 24:19–30

Cai X, Goh CJ, Mees AI (1998) Greedy heuristics for rapid scheduling of trains on a single track. IIE Trans 30:481–493

Carey M, Carville S (2003) Scheduling and platforming trains at busy complex stations. Transp Res Part A 37:195–224

Chvatal V (1985) Hamiltonian cycles. In: Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds) The traveling salesman problem—a guided tour of combinatorial optimization. Wiley, Chichester, pp 403–429

Cordeau JF, Toth P, Vigo D (1998) A survey of optimization models for train routing and scheduling. Transp Sci 32(4):380–404

Dessouky MM, Lu Q, Zhao J, Leachman C (2006) An exact solution to determine the optimal dispatching times for complex rail networks. IIE Trans 38:141–152

Eppstein D (2001) Bibliography on k shortest paths and other 'k best solutions' problems. http://www.ics.uci.edu/~eppstein/bibs/kpath.bib
Eppstein D (1998) Finding the k shortest paths. SIAM J Comput 28(2):652–673
Freling R, Lentink RM, Kroon LG, Huisman D (2005) Shunting of passenger train units in a railway station. Transp Sci 39(2):261–272
Hamacher HW, Queyranne M (1985) K-best solutions to combinatorial optimization problems. Ann Oper Res 4:123–143
He S, Song R, Chaudhry SS (2003) An integrated dispatching model for rail yards operations. Comp Oper Res 30:939–966
Lawler EL (1973) A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. Manage Sci 18:401–407
Libura M (1991) Sensitivity analysis for minimum weight base of a matroid. Control Cybern 20:7–24
Lübbecke ME, Zimmerman UT (2003a) Engine routing and scheduling at industrial in-plant railroads. Transp Sci 37(2):183–197
Lübbecke ME, Zimmerman UT (2003b) Computer aided scheduling of switching engines. In: Jäger W, Krebs HJ (eds) Mathematics—key technology for the future: joint projects between universities and industry, pp 690–702
Shier DR, Witzgall C (1980) Arc tolerances in shortest path and network flow problems. Networks 10: 277–291
Shier DR (1976) Iterative methods for determining the k shortest paths in a network. Networks 6:205–230
van der Poort ES, Libura M, Sierksma G, van der Veen JAA (1999) Solving the k-best travelling salesman problem. Comp Oper Res 26(4):409–425
Yen JY (1971) Finding the k shortest loopless paths in a network. Manage Sci 17:712–716
Zwaneveld PJ, Kroon LG, Romeijn HE, Salomon M, Dauzère-Pérès S, van Hoesel SPM, Ambergen HW (1996) Routing trains through railway stations: model formulation and algorithms. Transp Sci 30(3):181–194